Neural and Evolutionary Computation (NEC)

A1: Prediction with Back-Propagation and Linear Regression

Objective

Prediction using the following algorithms:

- **Back-Propagation** (**BP**), implemented by the student
- **Multiple Linear Regression** (**MLR**), using free software

Datasets

The predictions must be performed on three datasets:

1. File: *A1-turbine.txt*
    - 5 features: the first 4 are the input variables, the last one is the value to predict
    - 451 patterns: use the first 85% for training and validation, and the remaining 15% for test
2. File: *A1-synthetic.txt*
    - 10 features: the first 9 are the input variables, the last one is the value to predict
    - 1000 patterns: use the first 80% for training and validation, and the remaining 20% for test
3. Search a dataset from the Internet, with the following characteristics:
    - At least 6 features, one of them used for prediction
    - The prediction variable must take real (float or double) values; it should not represent a categorical value (that would correspond to a classification task)
    - At least 400 patterns
    - Select randomly 80% of the patterns for training and validation, and the remaining 20% for test; it is important to shuffle the original data, to destroy any kind of sorting it could have
    - In case of doubt on the suitability of the selected data, please ask the professor
    - Add to the documentation of this assignment the link to the source webpage

Procedure

- We want to simulate a real application, thus:
    - Apply the techniques of data preprocessing to the third dataset, the one you have found on the Internet; the provided datasets are already cleaned, no need to preprocess them. Basically, check for missing values, represent correctly categorical values, and look for outliers
    - Data normalization is necessary for BP training of all datasets
- You must find good values for all the parameters of BP: architecture of the network, learning rate and momentum, activation function, and number of epochs. Try different combinations from a certain set and use the expected error to select the best-

performing set of parameters. Once you have found those best parameters, apply them to the test set.

- Spend some time trying to automate the whole process, otherwise you will spend a lot of time repeating the same manual changes of the parameters

Implementation of BP

- Choose any programming language, but with the following restrictions:
    - Julia >= 1.8, Python >= 3.6, R >= 4.0, Matlab >= 2020a, Java >= 8
    - Libraries that implement neural networks and BP are forbidden
    - Libraries that implement the reading of the data and the scaling of the features are accepted
    - Plotting libraries are also accepted
- It is recommended to use a compiled programming language (Julia, C, C++, C#, Java or Ada) instead of interpreted ones (Python, Matlab, R) because BP requires a lot of calculations, which can take seconds or minutes with the former, and hours with the later ones.
- The implementation must be based on the algorithm and equations in document [G] of Unit 3 at Moodle
- The implementation must use the following variables to hold all the information about the structure of the multilayer neural network and the BP:
    - `L`: number of layers
    - `n`: an array with the number of units in each layer (including the input and output layers)
    - `h`: an array of arrays for the fields (h)
    - `xi`: an array of arrays for the activations (ξ)
    - `w`: an array of matrices for the weights (w)
    - `theta`: an array of arrays for the thresholds (θ)
    - `delta`: an array of arrays for the propagation of errors (Δ)
    - `d_w`: an array of matrices for the changes of the weights (δw)
    - `d_theta`: an array of arrays for the changes of the weights (δθ)
    - `d_w_prev`: an array of matrices for the previous changes of the weights, used for the momentum term ($\delta w^{(prev)}$)
    - `d_theta_prev`: an array of arrays for the previous changes of the thresholds, used for the momentum term ($\delta\theta^{(prev)}$)
    - `fact`: the name of the activation function that it will be used. It can be one of these four: sigmoid, relu, linear, tanh.
- For example, the weight $w_{ij}^{(L)}$ between unit `j` in layer `L-1` and unit `i` in layer `L` is accessed as `w[L][i,j]` in Julia and C#, or `w[L][i][j]` in C and Java
- The idea behind this structure is that the code must be able to deal with arbitrary multilayer networks. For example, a network with architecture 3:9:5:1 (4 layers, 3 input units, 1 output unit, and two hidden layers with 9 and 5 units, respectively), would have `n=[3; 9; 5; 1]`, and `xi` would be an array of length 4 (one component per layer), with `xi[1]` and array of real numbers of length 3, `xi[2]` and array of real numbers of length 9, `xi[3]` and array of real numbers of length 5, and `xi[4]` and array of real numbers of length 1. Similarly, `w[2]` would be an array `9x3`, `w[3]` an array `5x9`, and `w[4]` and array `1x5`; `w[1]` is not used.

- Additionally, the use of this structure, name conventions and array dimensions makes it easy to convert the equations into code

Training parameters and execution

- All the training parameters must be put in a text file. It must include:
    - Name of the data file
    - Number of training and test patterns
    - Information about the type of activation function
    - Number of layers
    - Number of units in each layer
    - Number of epochs
    - Learning rate and momentum
    - Optionally: information about the scaling method (normalization or standardization) of inputs and/or outputs, and in the case of normalization, the range of the normalized data
    - Optionally: the selected activation function (sigmoid, tanh, ReLU, etc.)
    - Optionally: name of output file(s)
- If compilation is needed, add a script for compilation, e.g., a file `compile.sh`, `compile.bat`, or a `Makefile`
- The execution should be something like this:
    - `./run_bp parameters_file.txt`
    - `./run_bp.exe parameters_file.txt`
    - `./run_bp.sh parameters_file.txt`
- Comment: for the search of the best-performing parameters you may need to work with other scripts and/or programs; that's acceptable, but the availability of the execution of BP using a parameters file is mandatory, even if this is not what you have used to obtain your results
- Plots of training and validation quadratic errors as a function of the epoch are useful to decide the number of epochs and to avoid over-fitting (over-training)

Multilinear regression

- There is no need to implement MLR, you may use any free tool you want. Anyway, I recommend using libraries from Julia, Python, R, Octave or Matlab, since they allow to use them with a simple script
- Other alternatives, like Weka, Excel and others require the use of a graphical interface, thus they are not easily automated and we don't recommend its usage

Evaluation of the results

- Although BP and MLR are based on the minimization of the mean squared error, it does not constitute a useful measure of the prediction error. We will use the mean absolute percentage error (MAPE), given by:

$$E(\%) = 100\ \frac{\sum_{\mu}|y^{\mu} - z^{\mu}|}{\sum_{\mu} z^{\mu}}$$

- Evaluate the previous measure (MAPE) on the Test and Validation sets for both BP and MLR, and for all the datasets
- The best way to visualize the results is with scatter plots of the prediction value y^{μ} compared with the real value z^{μ}. The closer the points are to the diagonal, the better the prediction. You must include them in the documentation

Delivery

- This assignment can be done alone or in pairs (groups of two)
- The delivery must be done in a compressed file (e.g., zip, rar, tgz) whose name should be of the form:
  - A1-Name1Surname1-Name2Surname2.zip
- The delivery must contain:
  - Files:
    - Source code of BP (avoiding intermediate files, e.g., *.o, *.class, etc)
    - Source code of MLR, if any
    - Parameters files (for the best-performing BP)
    - Script files
    - Results files for MLR and for your best-performing BP: CSV (or TXT tab separated) files with 2 columns, the real value z^{μ} and the prediction y^{μ} for the test sets
    - Other result files (e.g., weights and threshold of the best-performing trained networks, etc.)
  - Report in pdf:
    - Description of the implementation (languages, tools used, etc.)
    - Execution instructions
    - Implementation decisions
    - Description and link to the selected dataset
    - Predictions
    - Evaluation of the predictions: test error
    - Scatter plots of the prediction versus real value for both BP and MLR on the Test subsets
    - Discussion and interpretation of the results